CHAPTER 5

WRITING MORE EFFECTIVE AND EFFICIENT PROGRAMS

FOR-NEXT, STEP, DIM, GOSUB, RETURN, Arrays, and Nested Loops

By now you realize the power of a computer is in its capability to do many computations over and over on different data. While a great deal of detail and precision is required when writing a program, once written it can be used again and again. You can see that as problems increase in size and complexity, programming becomes more tedious and time consuming, especially if you are limited only to the keywords presented in Chapters 3 and 4. Fortunately, there are additional keywords:

- FOR-NEXT—for simplifying loops
- DIMENSION and subscripted variables—for processing data in tables (arrays of one or more dimensions)
- Predefine functions—for computing commonly used mathematical functions
- DEF—for defining your own functions
- GOSUB and RETURN—to allow the use of subroutines
- STOP—to terminate program execution anywhere in a program

SIMPLIFYING LOOPS USING FOR-NEXT

In Chapter 4, we saw that loops can be very useful when you have a series of statements you wish to repeat a number of times. BASIC provides two additional keywords that make some loops even easier to construct. They are **FOR-TO** and **NEXT.**

A FOR-NEXT loop always begins with a FOR-TO statement and always ends with a NEXT statement. The complete loop is comprised of all statements included between the FOR-TO and the NEXT statements.

Example:

```
45    FOR M = 1 TO 5
        .
        .
        .
75    NEXT M
```

This loop will consist of all statements, from statement number 45 through statement number 75, and it will be executed 5 times.

The FOR-TO statement specifies how many times the loop is to be executed. It must be the first statement in the loop. The FOR-TO statement has a numeric variable, called the running variable, whose value changes each time the loop is executed. The number of executions is determined by specifying the initial and final values for the running variable. In the example, M is the running variable; 1 is the initial value of M, and 5 is the final value of M. Each time through the loop, M is increased by 1. When M equals 5 the program exits the loop.

The NEXT statement consists of a statement number, followed by the keyword, NEXT, followed by a running variable name. This running variable must be the same as the running variable that appears in the corresponding FOR-TO statement.

FOR-TO Statement

A typical FOR-TO statement would look like this:

Example:

```
65  FOR M = 1 TO 36
```

In this example, M is the running variable. The first time the loop is executed, M will be assigned a value of 1. M will increase by 1 each time the loop is executed, until M has reached its final value of 36. The loop will be terminated once M has exceeded its final value of 36. The loop in the example will be executed 36 times.

The running variable will always increase by 1, if the FOR-TO statement contains no instructions telling it to do otherwise. However, we can increment the running variable by some value other than 1 if we wish. This can be done by the addition of a **STEP** clause to the FOR-TO statement.

Suppose we want to execute a loop 50 times, and we want the running variable to increase by 2 after each execution. We could write it this way:

Example:

```
65  FOR M = 1 TO 99 STEP 2
```

The running variable, M, would be assigned a value of 1 during the first pass; a value of 3 during the second pass; 5 during the third pass and so on, until the value of M was 99 during the 50th (final) pass.

The running variable does not have to be a positive integer value; it can be a negative or decimal value. Also, the running variable can be made to decrease with each execution of the loop. This is done by specifying a negative value in the STEP clause. The initial, final, and STEP values assigned to the running variable can be expressed as variables or expressions as well as numbers.

Examples:

```
20    FOR A = .5 TO 1.5 STEP .1
30    FOR B = C TO 0 STEP -1
40    FOR D = F1 TO F2 STEP F3
50    FOR E = G/10 TO (A+B)**3 STEP L+1
```

Some important points to know and remember when creating a FOR-TO-NEXT loop are:

- The loop begins with a FOR-TO statement and ends with a NEXT statement.
- The same running variable name must be used in the FOR-TO and NEXT statements.
- The running variable can appear in a statement inside the loop, but its value cannot be changed.
- The running variable will be incremented by 1 unless otherwise specified by a STEP clause.
- If the initial and final values of the running variable are equal, and the step size is nonzero, the loop will be executed once.
- There are three conditions under which a loop will not be executed at all.
  1. The initial and final values of the running variable are equal and the step size is zero.
  2. The final value of the running variable is less than the original value, and the step size is positive.
  3. The final value of the running variable is greater than the original value, and the step size is negative.
- Control can be transferred out of a loop but not in. (The transfer out can be done by using a GOTO, an ON-GOTO, or an IF-THEN statement.)

Examine the following loop and see how it conforms to the points just listed.

Example:

```
600   FOR A = 0 TO 1.6 STEP .2
        .
        .
        .
650   LET X = A + B
660   IF X > X1 THEN 900
        .
        .
        .
700   NEXT A
        .
        .
        .
900   PRINT A,B,X
```

This example shows the use of the running variable (A) within the loop (line number 650). The statement in line number 660 will cause control to be transferred outside the loop, if the value of X is greater than the value of Xl. Also, the running variable (A) that appears in the NEXT statement is the same as the running variable in the FOR-TO statement. These must be the same or the loop won't work. The step clause specifies that A is to be increased by .2 each time the loop is executed. If X does not exceed Xl the loop will be executed 9 times.

The following mortgage amortization program contains an example of the FOR-NEXT loop structure.

Example:

```
10    REMARK MONTHLY MORTGAGE AMORTIZATION
20    PRINT ''ENTER MONTHLY PAYMENT (D) LOAN AMOUNT (B)'';
30    PRINT ''INTEREST RATE (I) NUMBER OF MONTHS (N).''
40    INPUT D,B,I,N
50    PRINT ''MONTH'';'' PAYMENT'';'' LOAN BALANCE'',''PRINCIPLE'',
60    PRINT ''INTEREST''
70    LET R = I/12
80    FOR M = 1 to N
90    LET A = B*R
100   LET P = D - A
110   LET B = B - P
120   PRINT M;D;B,P,A
130   NEXT M
140   PRINT
150   PRINT ''WITH ONE FINAL PAYMENT OF'';B
999   END
```

The loop in this example is comprised of lines 80 through 130. Line 80 sets the initial value of M to 1 for the first execution of the loop. Lines 90 through 120 perform the calculations and print the results. Once the PRINT statement in line 120 has been executed, the NEXT statement in line 130 directs the computer to start the loop all over again. The loop will continue until the number of times it has been executed is equal to N (line 80). N is the number of months of the loan.

This example shows the use of a single loop structure using the FOR-TO. . .NEXT statements. It is also possible to have a loop within a loop. These are called *nested loops.*

Nested Loops

In addition to the rules which apply to single loops, the following rules apply to nested loops:

- Each nested FOR-NEXT loop must begin with its own FOR-TO statement and end with its own NEXT statement.
- An outer loop and an inner (nested) loop cannot have the same running variable.
- Each inner (nested) loop must be completely embedded within an outer loop, the loops cannot overlap.
- Control can be transferred from an inner loop to a statement in an outer loop or to a statement outside of the entire nest. However, control cannot be transferred to a statement within a nest from a point outside the nest.

The following example shows the structure of a nested loop.

Example:

```
65    FOR X = 1 TO 10
      .
      .
      .
90    FOR Y = X TO Z
      .
      .
      .
115   NEXT Y
      .
      .
      .
140   NEXT X
```

The inner loop (statements 90 through 115) is completely embedded within the outer loop (statements 65 through 140). Each loop begins and ends with its own FOR-TO and NEXT statements, and each loop has its own running variable. You will notice the running variable of the outer loop (X) is used as the initial value for the running variable of the inner loop (Y). This is allowed since the value of X is not changed within the inner loop.

By using nested loops, you are able to perform repeated sets of instructions within another set of instructions.

Example:

```
10    REM THIS PROGRAM WILL COMPUTE THE SQUARE, CUBE,
20    REM AND 4TH POWER OF THE NUMBERS 1 TO 10
30    FOR X = 1 TO 10
40    PRINT X,
50    LET A = X
60    FOR Y = 1 TO 3
70    LET A = A*X
80    PRINT A,
90    NEXT Y
100   PRINT " "
110   NEXT X
120   END
RUN
 1            1            1            1
 2            4            8            16
 3            9            27           81
 4            16           64           256
 5            25           125          625
 6            36           216          1296
 7            49           343          2401
 8            64           512          4096
 9            81           729          6561
 10           100          1000         10000
```

The outer loop (lines 30-1 10) will be executed 10 times, while the inner loop (lines 60-90) will be executed 3 times for each time the outer loop is executed. This means the inner loop is executed a total of 30 times.

WORKING WITH ARRAYS

In BASIC, we have the capability to store and reference data elements in lists or tables. These are called *arrays.* An entire array is assigned one name (e.g., D), and yet, we can refer to any element in the array by using a subscripted variable. If D is such an array, then D(l), read "D sub one, "

is the first element in array D. The value in parentheses is called a subscript. It indicates the relative position of a given element in an array. For example, LET D(3) = D(1) + D(2) is a BASIC statement which adds the first two numbers of array D and puts the sum in the third element of array D.

Before elements in arrays can be used, a method is needed to define the array. The **DIM** (DIMENSION) statement is used for this. It names the array and reserves memory space. For example, DIM D(15) would reserve space for 15 data elements with the name, D.

When subscripted variables are used, their corresponding values must be read and stored in memory during program execution. Examine the following example (one-dimensional array) and see how the data is stored in the computer's memory when the program statements are executed.

Example:

```
          .
          .
          .
750       FOR L = 1 TO 5
760       READ S(L)
770       NEXT L
780       DATA 72,74,69,70,71
          .
          .
          .
999       END
```

S(1)	S(2)	S(3)	S(4)	S(5)
72	74	69	70	71

In this example, the loop will be executed five times. The variable used to control the loop is also used as the subscript for S in the READ statement. On the first pass through the loop, the subscript L is 1, the value of L during the first execution of the loop. On the second pass, the subscript will be 2, and so on, until the value of L is equal to 5.

Subscripted variables can also be used to identify elements in tables (two-dimensional arrays) but it takes two subscripts, one to specify the row and a second one to specify the column. For example, S(2,5) would specify the location of the value in the second row, fifth column. Figure 5-1 is an example of a two-dimensional array.

DIM (DIMENSION) Statement

As stated earlier a method is needed to specify the size of an array. The BASIC programming language automatically assigns 11 elements to every

one-dimensional array and 121 elements (11 rows and 11 columns) to every two-dimensional array appearing in a program.

Larger arrays may be used. However, the size of each must be defined; that is, you must specify the maximum number of elements in each. The following example shows how a DIM statement is constructed.

Example:

```
10 DIM A(25), B(50,5)
```

This DIM statement would reserve memory space for an array named A with 25 elements, and an array named B with 50 rows and 5 columns. On some computers the first element in a one-dimensional array is referenced with subscript 0, and in a two-dimensional array by subscripts, 0, 0. If that is the case on your computer, you would set the upper limits at one less than the number of elements you need.

When the BASIC interpreter encounters a DIM statement such as the one above, it reserves an area in memory for arrays A and B made up of 25 and 250 elements respectively.

If an array requires less storage space than is automatically reserved by the BASIC interpreter it need not be defined by a DIM statement. However, by using a DIM statement only the space actually needed will be reserved.

Arrays

Some important things to remember about arrays:

- DIMENSION statements are used to define arrays.
- The elements in an array can be either numeric quantities or strings. However, all of the elements in a given array must be the same type (all numeric or all string).
- An array that contains numeric elements must be named with a single letter.
- A string array is referred to with a letter followed by a dollar sign.
- Elements in a one-dimensional array are referenced by the array name followed by a subscript in parentheses, A(l).
- Elements in a two-dimensional array, *matrix*, are referenced by the name followed by two subscripts in parentheses; the first references the row, the second the column, A(2,3).
- Subscripts may be a numeric-constant or a numeric-variable.
- Each array name in a program must be unique. However, an array and an ordinary variable can have the same name. Duplicating array names and variable names could be logically confusing; therefore, it is not a recommended practice.

Many computers have a special set of instructions called **MAT** instructions for working with matrices. By using a single MAT instruction, matrices may be defined, added, subtracted, multiplied, read, and manipulated in a variety of ways. Any of these operations may be done with FOR-NEXT loops; however, the MAT statement, if available, makes it easier. See Appendix II for examples and refer to the user's manual for your specific computer.

Sample Problem Using Nested Loops and a Matrix

Suppose you wanted to write a program to compute your career sea pay based on your paygrade and years of sea duty. You would need a two-dimensional array (matrix) to store the data. Figure 5-1 shows the table of values needed to determine sea pay.

Examine the following program and see how the matrix is constructed. The program contains nested loops (lines 20-60) which are used to read the values into the matrix. The outer loop sets up the row portion, G, which represents paygrade. The inner loop sets up the column portion, S, which represents the years of sea duty.

ARRAY P
(TWO-DIMENSIONAL ARRAY)

ROW (G) PAYGRADE		COLUMN (S) YEARS SEA DUTY 1	2	3	4	5	6	7	8	9	10	11	12
E-4	1	60	125	160	175	175	175	175	175	175	175	175	175
E-5	2	70	140	175	185	190	205	220	220	220	220	220	220
E-6	3	135	170	190	210	215	225	235	245	255	255	255	255
E-7	4	145	215	235	255	260	265	265	270	275	280	300	310
E-8	5	180	225	255	265	270	280	285	290	300	310	310	310
E-9	6	195	235	265	280	290	310	310	310	310	310	310	310

Figure 5-1.—Sea-pay table.

Example:

```
10    DIM P(6,12)
20    FOR G = 1 TO 6
30    FOR S = 1 TO 12
40    READ P(G,S)
50    NEXT S
60    NEXT G
70    PRINT ''SEA PAY CALCULATION PROGRAM (E4-E9)''
80    PRINT ''INPUT WHOLE NUMBERS ONLY''
90    PRINT ''WHAT IS YOUR PAYGRADE''
100   INPUT G
110   LET G = G - 3
120   PRINT ''HOW MANY YEARS SEA DUTY (1-12)''
130   INPUT S
140   PRINT ''YOUR SEA PAY SHOULD BE $'';P(G,S)
150   DATA 60,125,160,175,175,175,175,175,175,175,175,175
160   DATA 70,140,175,185,190,205,220,220,220,220,220,220
170   DATA 135,170,190,210,215,225,235,245,255,255,255,255
180   DATA 145,215,235,255,260,265,265,270,275,280,300,310
190   DATA 180,225,255,265,270,280,285,290,300,310,310,310
200   DATA 195,235,265,280,290,310,310,310,310,310,310,310
210   END

RUN
SEA PAY CALCULATION PROGRAM (E4-E9)
INPUT WHOLE NUMBERS ONLY
WHAT IS YOUR PAYGRADE
?8
HOW MANY YEARS SEA DUTY (1-12)
?7
YOUR SEA PAY SHOULD BE $  285
```

As seen in the output from this program, an E-8 with 7 years sea duty would receive $285.00 sea pay. Try the program and see what your sea pay would be.

The paygrade (4-9) is entered (line 100). Before it can be used as a subscript to determine row number, we subtract 3 (line 110). This makes it correspond to row number 1-6. Next, years of sea duty (1-12) are entered to be used as the subscript for column number. Then the PRINT statement (line 140) prints the corresponding value of the coordinates G and S from the matrix named P.

USING PREDEFINED FUNCTIONS

Some of the more commonly used mathematical functions have been predefined in the BASIC language. They were presented in Chapter 2 and are listed in Appendix I. To use them, all you have to do is specify the function and provide an argument (the number or variable, on which the function is to be executed).

Examples:

Calculate and Print the Square Root of 16

```
20 PRINT SQR (16)
```

Calculate the Absolute Value of X–Y

```
30 LET Z = ABS(X – Y)
```

If X = 10 and Y = 4, then X - Y = 6 and the absolute value is 6.

If X = 4 and Y = 10, then X-Y = –6 and the absolute value is also 6.

DEFINING YOUR OWN FUNCTIONS

In addition to the predefine functions, BASIC allows you to define your own functions within a program. The statement for defining functions is the **DEF** (DEFINE) statement.

The DEF statement consists of a statement number, the keyword DEF and the function definition. The function definition consists of the function name, followed by an equal sign, followed by a constant, variable, or expression. If the function requires an argument, then it must appear immediately after the function name, enclosed in parentheses. The following example shows how a function to convert Fahrenheit to Celsius could be defined.

Example:

```
40 DEF FNC(F) = (F – 32)*5/9
```

Both numeric and string functions may be defined with the DEF statement. Numeric functions return numeric values and string functions return string values. Numeric function names must consist of three letters, the first two must be **FN,** followed by any single letter of the alphabet (A-Z). Therefore,

as many as 26 separate numeric functions can be defined in a single program (FNA, FNB,...FNZ).

String functions must consist of three letters followed by a dollar sign. Like numeric functions, the first two letters must be FN. Up to 26 separate string functions may be defined in a single program (FNA$, FNB$,...FNZ$). Numeric and string functions having the same three letters (FNA and FNA$) are considered as two different functions and may appear in the same program.

Some important things to remember about user defined functions are:

- A function definition statement must have a lower numbered line than that of the first reference to the function.
- The expression in a DEF statement is evaluated only when the defined function is referenced.
- If the execution of a program reaches a line containing a DEF statement, it proceeds to the next line with no other effect.
- A function definition can reference other defined functions, but not itself.
- A function may be defined only once in a program.
- Predefine functions may be used in arguments of user defined functions.
- Subscripted variables are not permitted as arguments in a function definition.

The following example shows a user defined function to calculate the area of a circle.

Example:

```
10  DEF FNA(R) = 3.1416*R**2
```

You can define your own functions, include them, and use them in your program. The following program shows the use of this function in a program to compute the areas of any number of circles.

Example:

```
10   DEF FNA(R) = 3.1416*R**2
20   PRINT "THIS PROGRAM WILL GIVE THE AREAS OF ANY NUMBER"
30   PRINT "OF CIRCLES THAT YOU SPECIFY"
40   PRINT "HOW MANY CIRCLES DO YOU WANT?"
50   INPUT N
60   FOR X = 1 TO N
70   PRINT "ENTER RADIUS"
80   INPUT Y
90   PRINT "CIRCLE # ";X,"RADIUS = ";Y, "AREA = ";FNA(Y)
100  NEXT X
110  END

RUN
```

```
THIS PROGRAM WILL GIVE THE AREA OF ANY NUMBER OF CIRCLES
THAT YOU SPECIFY
HOW MANY CIRCLES DO YOU WANT?
?5
ENTER RADIUS
?6
CIRCLE # 1    RADIUS = 6  AREA = 113.0976
ENTER RADIUS
?3
CIRCLE # 2    RADIUS = 3  AREA = 28.2744
ENTER RADIUS
?9
CIRCLE # 3    RADIUS = 9  AREA = 254.4696
ENTER RADIUS
?1
CIRCLE # 4    RADIUS = 1  AREA = 3.1416
ENTER RADIUS
?7
CIRCLE # 5    RADIUS = 7  AREA = 153.9384
```

The function to compute the area of a circle is defined in line 10. The PRINT statement, line 90, references the defined function to print the area of the circle. The variable name (R) used in the DEF statement is not the same as the one (Y) used in the PRINT statement where the function is referenced, rather it corresponds to the variable name used in the INPUT statement, line 80.

CONSTRUCTING AND USING SUBROUTINES

Like functions, subroutines are designed so they can be used over and over within a program, or so they can be inserted easily into other programs.

A *subroutine is* defined as a small program within another program. It does not have to be given a name or begin with a particular keyword. Subroutines can be used when sets of instructions are to be performed several

times in one or more programs. They can also be useful when more than one programmer is working on a program. Each programmer can be assigned a portion of the program to write, and that portion can be written as a subroutine. When all portions of the program have been written, they can be put together and referenced as subroutines in the main program. This reduces the possibility of statements written by one programmer conflicting with the statements written by another programmer.

To execute a subroutine, you must transfer control to the subroutine by using the keyword, **GOSUB.** Once executed, the subroutine transfers control back to the statement immediately following the GOSUB statement.

GOSUB and RETURN Statements

The **GOSUB** statement is used to transfer control to a subroutine. It is made up of a statement number followed by the keyword GOSUB and the number of the first statement in the subroutine.

The **RETURN** statement is used to transfer control back to the main program. It must be the last statement in the subroutine. The RETURN statement consists of a statement number followed by the keyword RETURN. When the RETURN statement is executed, it transfers control back to the main program to the statement immediately following the GOSUB statement. The structure is as follows:

```
30      GOSUB 200
40      .
        .
        .
200     LET X = Y + Z
        .
        .
        .
270     RETURN
```

Line 30 transfers control to line 200, the first statement of the subroutine. Line 270, the last statement of the subroutine, returns control to line 40, the line immediately following the GOSUB.

The following example shows the use of the GOSUB and RETURN statements in transferring control to a subroutine and returning control back to the main program.

Example:

```
10    REM THIS PROGRAM COMPUTES INTEREST
20    REM EARNED ON SAVINGS
30    PRINT ''ENTER AMOUNT SAVED 1000-4999''
40    INPUT X
50    ON X/1000 GOTO 100,120,140,140
51    PRINT ''VALUES ENTERED OUT OF RANGE, TRY AGAIN''
52    GOTO 30
60    PRINT ''ANNUAL INTEREST ON $ '';X;''IS $ '';I
70    GOSUB 160
80    IF X$ = ''Y'' THEN 30
90    STOP
100   LET I = .04*X
110   GOTO 60
120   LET I = .045*X
130   GOTO 60
140   LET I = .05*X
150   GOTO 60
155   REM SUBROUTINE TO ASK IF MORE DATA
160   PRINT ''ENTER Y IF MORE DATA''
170   PRINT ''ENTER N IF NO MORE DATA''
180   INPUT X$
190   RETURN
999   END

RUN
ENTER AMOUNT SAVED
?1999
ANNUAL INTEREST ON $    1999 IS $    79.96
ENTER Y IF MORE DATA
ENTER N IF NO MORE DATA
?Y
ENTER AMOUNT SAVED
?2000
ANNUAL INTEREST ON $    2000 is $    90
ENTER Y IF MORE DATA
ENTER N IF NO MORE DATA
?N
```

The GOSUB statement in line 70 transfers control to the subroutine, line 160, which prints a prompt to the user. At this point, the subroutine gives the user the option, either to continue running the program or to terminate it. After a string value is entered at line 180, the RETURN statement, line 190 returns control to the main program at line 80. This line tests the value of the string variable, X$. If the value equals Y, control is transferred to line 30; if the value is equal to N, the program will STOP.

STOP Statement

The **STOP** statement, line 90, terminates execution of the program. Although the STOP statement terminates execution of the program, it does not replace the END statement. Remember, the END statement has two functions: to terminate execution of the program, and to indicate there are no more instructions for the BASIC interpreter to translate. Therefore you must include an END statement in every program, regardless of how many STOP statements you use. STOP statements may appear anywhere you need them in a program, and you can use as many as you want.

Summary

Constructing loops is made easier by using the FOR-NEXT loop structure. A FOR-NEXT loop always begins with a **FOR-TO** statement and ends with a **NEXT** statement. **A** numeric variable, called a running variable is used to control the number of times a loop is executed. The running variable used in the NEXT statement must be the same as the one used in the FOR-TO statement. The value of the running variable is incremented by one each time the loop is executed, unless a **STEP** clause is used to alter this. Control can be transferred out of a loop but not into one.

A loop may have another loop inside it. This is called a *nested loop.* Each nested loop must be completely embedded within the outer loop. They cannot overlap. Each nested loop must begin with its own FOR-TO statement and end with its own NEXT statement. Control cannot be transferred into a nested loop from a point outside the nest.

When working with *one- or two-dimensional arrays* you may reference any element in them by using a subscripted variable. The **DIM** (DIMENSION) statement is used to define the size of an array. It reserves space in the computer's memory for the specified number of elements.

Each array must be assigned a unique name, using either a numeric-variable name or a string-variable name. An array may contain either numeric or string data; however, all the elements in a given array must be of the same type (all numeric or all string).

Some of the more common mathematical functions have been predefine by the BASIC language; however, there are times when you may need to define your own. This can be done by using the **DEF** (DEFINE) statement.

A *subroutine is* a small program inside a larger program. It is useful when sets of instructions are to be performed several times in one or more programs, or when several programmers are working on the same program. A subroutine is executed by transferring control to the subroutine using a **GOSUB** statement with the statement number of the 1st statement in the subroutine. It is terminated by the keyword **RETURN.** When a RETURN

statement is executed, control is transferred from the subroutine back to the main program to the statement immediately following the GOSUB statement.

Execution of a program may be terminated anywhere in a program by the use of **STOP** statements.

CHAPTER 5

EXERCISES

1. Using a FOR-NEXT loop, write a program to print all the even numbers beginning with 20 and ending with 40.

2. Write a program that will compute the amount accumulated if you start with a penny and double it every day for 30 days. Print the total for each day.

3. Write a program that will read the following names into an array, then list them in reverse order. Carol, Chuck, Fred, Jane, John.

4. Write a program, including a DIM statement, to construct a matrix for 9 golfers with 4 games each. Include the capability to select and print a specific golfer number and a specific game. The scores for the players are:

	Game Number			
Golfer Number	1	2	3	4
1	69	72	70	75
2	73	72	74	70
3	71	75	69	73
4	70	74	72	71
5	69	68	70	72
6	75	77	73	70
7	68	66	70	72
8	70	73	71	69
9	76	71	74	72

5. Write a program containing a user defined function to compute the sale price for any piece of merchandise when given the original price; use 20% as the rate of discount.

6. Write a program containing a subroutine to calculate average miles per gallon.

CHAPTER 5

EXERCISE SOLUTIONS

The following programs present possible solutions to the exercises.

1. 10 FOR X = 20 TO 40 STEP 2

   20 PRINT X

   30 NEXT X

   40 END

   RUN

   20

   22

   24

   26

   28

   30

   32

   34

   36

   38

   40

2.
```
10 LET Y = .01
20 FOR X = 1 TO 30
30 LET Y = Y*2
35 PRINT Y
40 NEXT X
50 END

RUN

 2.00000000E-02
 4.00000000E-02
 8.00000000E-02
 .16
 .32
 .64
 1.28
 2.56
 5.12
 10.24
 20.48
 40.96
 81.92
 163.84
 327.68
 655.36
 1310.72
 2621.44
 5242.88
 10485.76
 20971.52
 41943.04
 83886.08
 167772.16
 335544.32
 671088.64
 1342177.28
 2684354.56
 5368709.12
 10737418.24
```

NOTE: Small numbers are represented by E notation.

3.
```
5  DIM A$(5)
10 DATA "CAROL"
20 DATA "CHUCK"
30 DATA "FRED"
40 DATA "JANE"
50 DATA "JOHN"
60 FOR X = 1 TO 5
70 READ A$(X)
80 NEXT X
90 FOR Y = 5 TO 1 STEP -1
100 PRINT A$(Y)
110 NEXT Y
120 END

RUN

JOHN
JANE
FRED
CHUCK
CAROL
```

4.
```
10 DIM G(9,4)
20 FOR P = 1 TO 9
30 FOR S = 1 TO 4
40 READ G(P,S)
50 NEXT S
60 NEXT P
```

```
70  PRINT "TOURNAMENT GOLF SCORES"
80  PRINT "ENTER PLAYER NUMBER (l-9)"
90  INPUT P
100 PRINT "ENTER GAME NUMBER (1-4)"
110 INPUT S
120 PRINT "PLAYER #";P;"SCORE FOR GAME #";S;"IS";G(P,S)
130 DATA  69,72,70,75,73,72,74,70,71,75,69,73
140 DATA  70,74,72,71,69,68,70,72,75,77,73,70
150 DATA  68,66,70,72,70,73,71,69,76,71,74,72
999 END

RUN

TOURNAMENT GOLF SCORES
ENTER PLAYER NUMBER (1-9)
?6
ENTER GAME NUMBER (1-4)
?2
PLAYER #6 SCORE FOR GAME #2 IS 77
```

5.
```
10 DEF FNP(C) = C – (.20*C)
20 PRINT "ENTER ORIGINAL PRICE"
30 INPUT Z
35 IF  Z = 0 THEN 60
40 PRINT "SALE PRICE IS $ ";FNP(Z)
50 GOTO 20
60 END

RUN

ENTER ORIGINAL PRICE
?15
SALE PRICE IS $ 12
```

6.
```
10 PRINT "ENTER MILES AND GALLONS"
20 INPUT M,G
30 GOSUB 60
40 PRINT "AVERAGE MILES PER GALLON IS";A
50 STOP
60 LET A = M/G
70 RETURN
99 END

RUN

ENTER MILES AND GALLONS
?250,10
AVERAGE MILES PER GALLON IS 25
```